Advanced NuSMV usage

Cyrille Artho and Roberto Guanciale
KTH Royal Institute of Technology, Stockholm, Sweden
School of Electrical Engineering and Computer Science
Theoretical Computer Science
artho@kth.se

2018-04-23

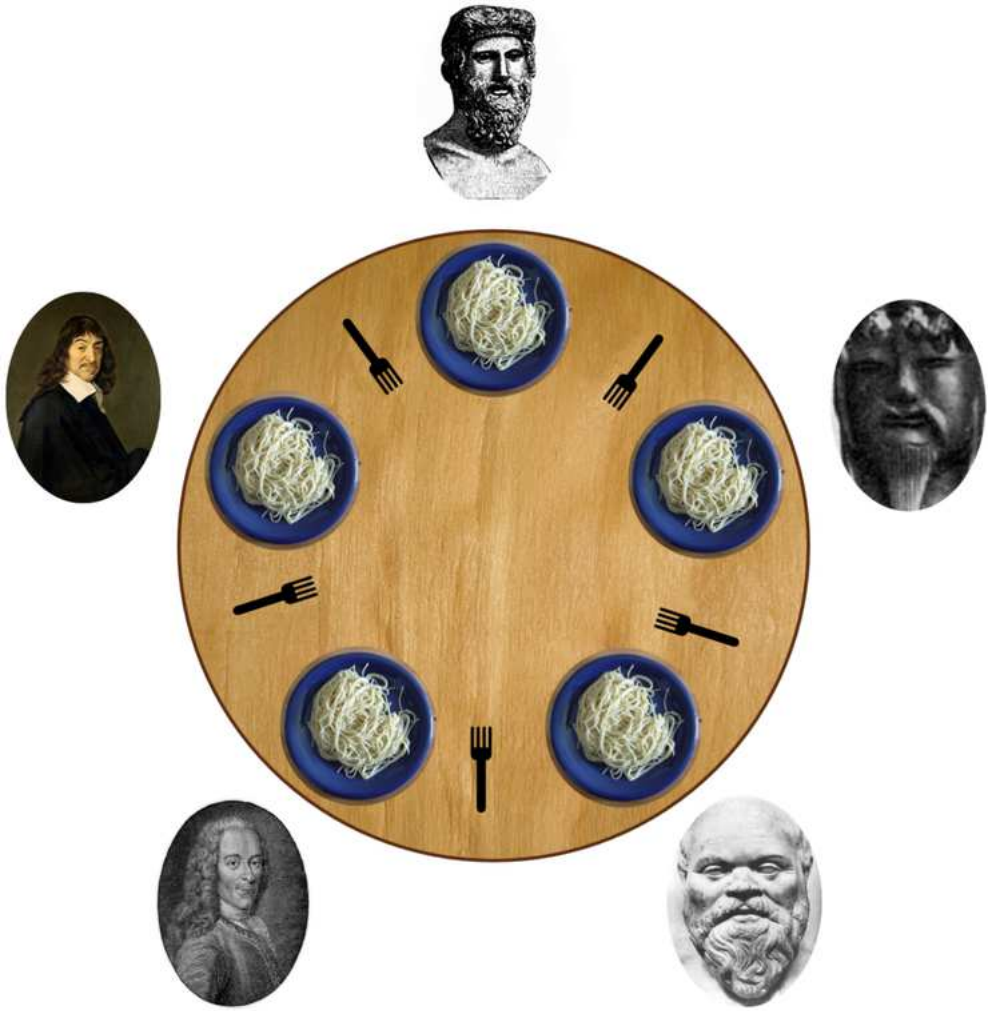
Outline

1. Bounded Model Checking, SAT solving.
2. Model validation.
3. Putting things together
   (a) Counter-example-guided abstraction refinement.
   (b) SLAM.
4. NuSMV in the real world.
5. Assignment: Elevator.

Advanced modeling: parametrized modules

- Instantiate multiple modules with different settings.
- Same transition rules for each instance.
- Avoid „copy pasta”.

Example: Dining Philosophers



- Five philosphers.
  - → Think eat.
  - → Need forks to eat.
- Five plates.
- Five forks.
  - → Exclusive access.
  - → Atomic access.

Can we get mutual exclusion without starvation?

Main module

```
MODULE main
VAR
  fork: array 1..5 of 0..5; -- 0: unused; > 0: held by X
  phil1: process philosopher(1, fork[1], fork[2]);
  phil2: process philosopher(2, fork[2], fork[3]);
  phil3: process philosopher(3, fork[3], fork[4]);
  phil4: process philosopher(4, fork[4], fork[5]);
  phil5: process philosopher(5, fork[5], fork[1]);
DEFINE
  available := 0;
ASSIGN
  init(fork[1]) := available;
  init(fork[2]) := available;
  init(fork[3]) := available;
  init(fork[4]) := available;
  init(fork[5]) := available;
```

Philosopher module

```
MODULE philosopher(id, leftFork, rightFork)
DEFINE
  owned := id;
  available := 0;
VAR
  state: {think, eat, done};
ASSIGN
  init(state) := think;
  next(state) := case
    (state = think) & (leftFork = owned) &
      (rightFork = owned): eat;
    (state = eat): {eat, done};
    (state = done): think;
   TRUE: think; -- (state = think) but forks not both taken
  esac;
```

Avoid copying module definition by using parameters.

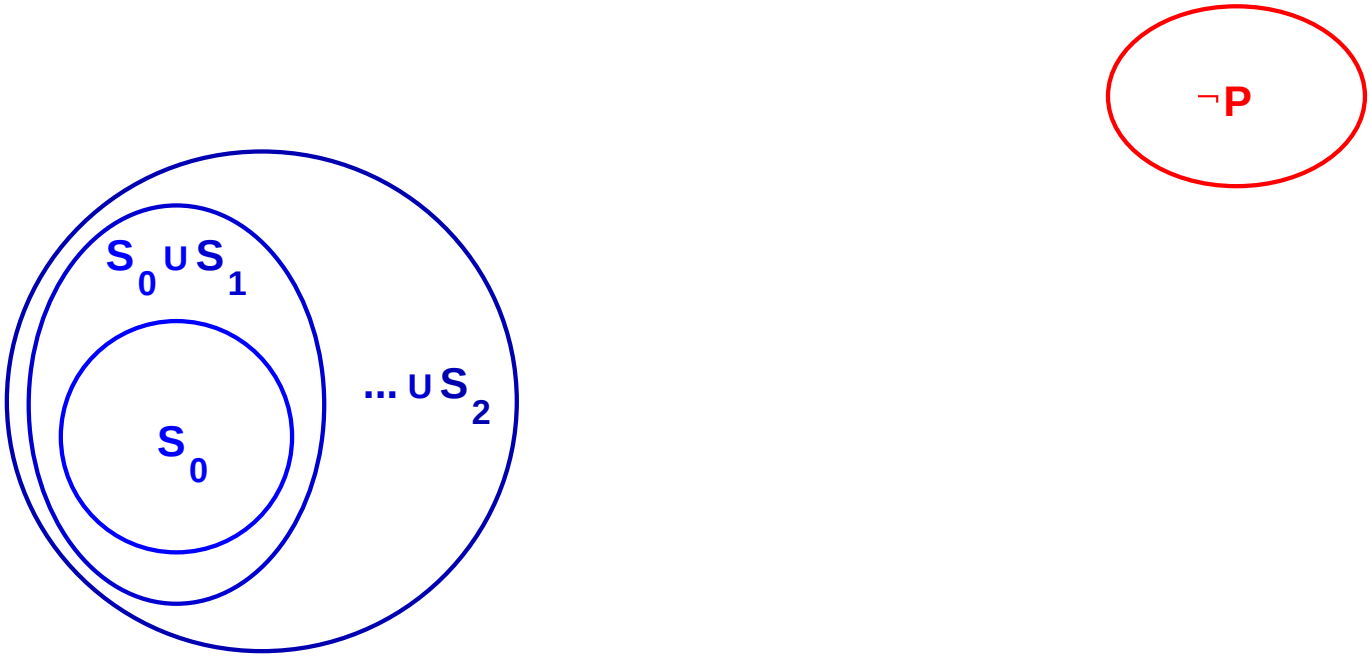
Fork transitions, properties

```
  next(leftFork) := case
    (state = think) & (leftFork = available):
      {available, owned};
    state = done: available;
    TRUE: leftFork;
  esac;
  next(rightFork) := case
    (state = think) & (rightFork = available):
      {available, owned};
    state = done: available;
    TRUE: rightFork;
  esac;
SPEC   AG ((state = eat) ->
           ((leftFork = owned) & (rightFork = owned)))
SPEC   AG EF (state = eat);
```

Which properties hold?

Bounded model checking

- Symbolic model checking is efficient if formula can be compressed.
- Good variable order is not always possible.
- Another approach: Expand the state transition system for k steps:



Use solver to check system states after k steps.

SAT solving

- Satisfiability (SAT) problem is NP-complete.
- SAT solvers are efficient solvers for logical formulas.
- Intelligent pre-processing and state space search take advantage of structure in formula.
- Does not work against worst case, but real formulas have internal structure.
- Expanded state transition system can be expressed in propositional logic and solved by SAT solver.

Bounded model checking in NuSMV

```
MODULE main -- from the NuSMV tutorial
VAR
  y : 0..15;
ASSIGN
  init(y) := 0;
TRANS
case
  y=7: next(y)=0;
  TRUE : next(y) = ((y + 1) mod 16);
esac
LTLSPEC G (y=4 -> X y=6)
```

Running NuSMV in BMC mode

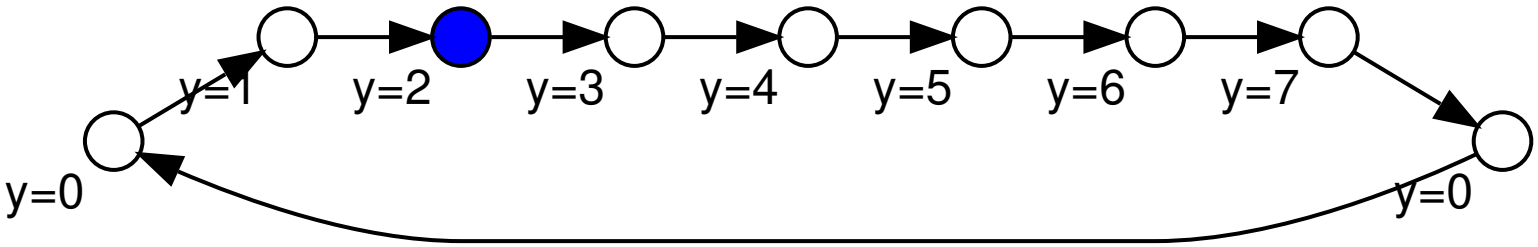
```
NuSMV -bmc bmc_tutorial.smv
-- no counterexample found with bound 0
-- no counterexample found with bound 1
-- no counterexample found with bound 2
-- no counterexample found with bound 3
-- no counterexample found with bound 4
-- specification  G (y = 4 ->  X y = 6) is false
-- as demonstrated by the following execution sequence
Trace Description: BMC Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    y = 0
  -> State: 1.2 <-
    y = 1
...
  -> State: 1.6 <-
    y = 5
```

Liveness properties with BMC

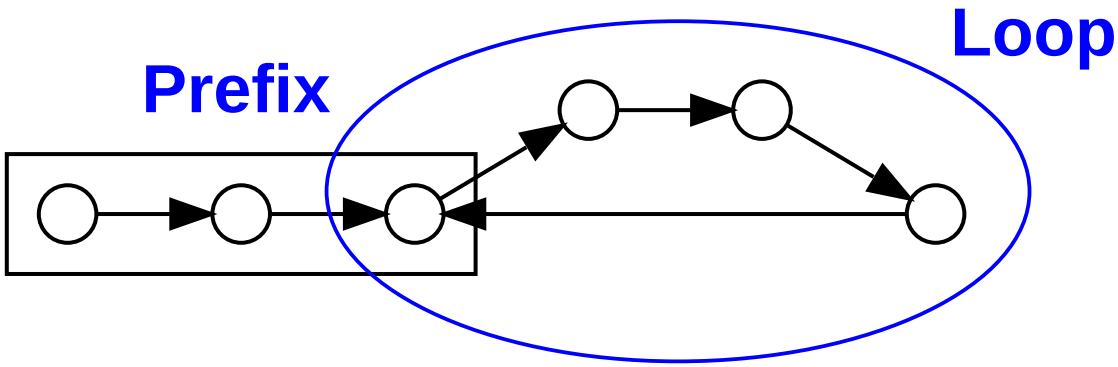
```
LTLSPEC !G F(y = 2)
NuSMV -bmc bmc_tutorial.smv
-- no counterexample found with bound 0
...
-- no counterexample found with bound 7
-- specification !( G ( F y = 2)) is false
-- as demonstrated by the following execution sequence
Trace Description: BMC Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 1.1 <-
y = 0
-> State: 1.2 <-
y = 1
...
-> State: 1.9 <-
y = 0
```

Counterexamples for liveness properties

Previous example:



General case:



How large does the bound k have to be for BMC?

Model validation

Model

Real system

Is the model adequate?

Correct but flawed models

Vacuity: $a \rightarrow b$ holds, but a is never true.
Entire property holds for the wrong reason („antecedent failure”)!

Solution: Check that a holds in at least some states.

Zeno-timelocks: Model executes infinitely fast.
No way for real system to fulfill property!

Solution: Use simulation mode to get example traces,
study how model reacts.

Model validation

1. Check the reachability of desired states.
   Use simulation mode or define additional liveness properties.

2. Negate the specification.
   Now there should be paths that fulfill the desired property.
   Study the result and see if it makes sense.

3. Add additional simple properties as sanity checks.

From: Artho, Hayamizu, Ramler, Yamagata: With an Open Mind: How to Write Good Models.

Simulating models with NuSMV

Key simulation commands:

`NuSMV > go`	Load and prepare model
`NuSMV > pick_state -i`	Pick initial state interactively
`NuSMV > simulate -p -i`	Simulate (interactively), print progress; control-C exits
`NuSMV > show_trace -v`	Show trace
`NuSMV > quit`	Exit

Back to vending machine

```
-> State: 1.1 <-
  choice = TRUE
  payment = TRUE
  acc_payment = FALSE
  state = ready
  release_item = FALSE
-> State: 1.2 <-
  state = expect_payment
-> State: 1.3 <-
  acc_payment = TRUE
-> State: 1.4 <-
  state = dispense_item
  release_item = TRUE
```

```
-> State: 1.5 <-
  acc_payment = FALSE
  state = ready
  release_item = FALSE
-> State: 1.6 <-
  state = expect_payment
-> State: 1.7 <-
  acc_payment = TRUE
-> State: 1.8 <-
  state = dispense_item
  release_item = TRUE
```

Usage of NuSMV in the real world

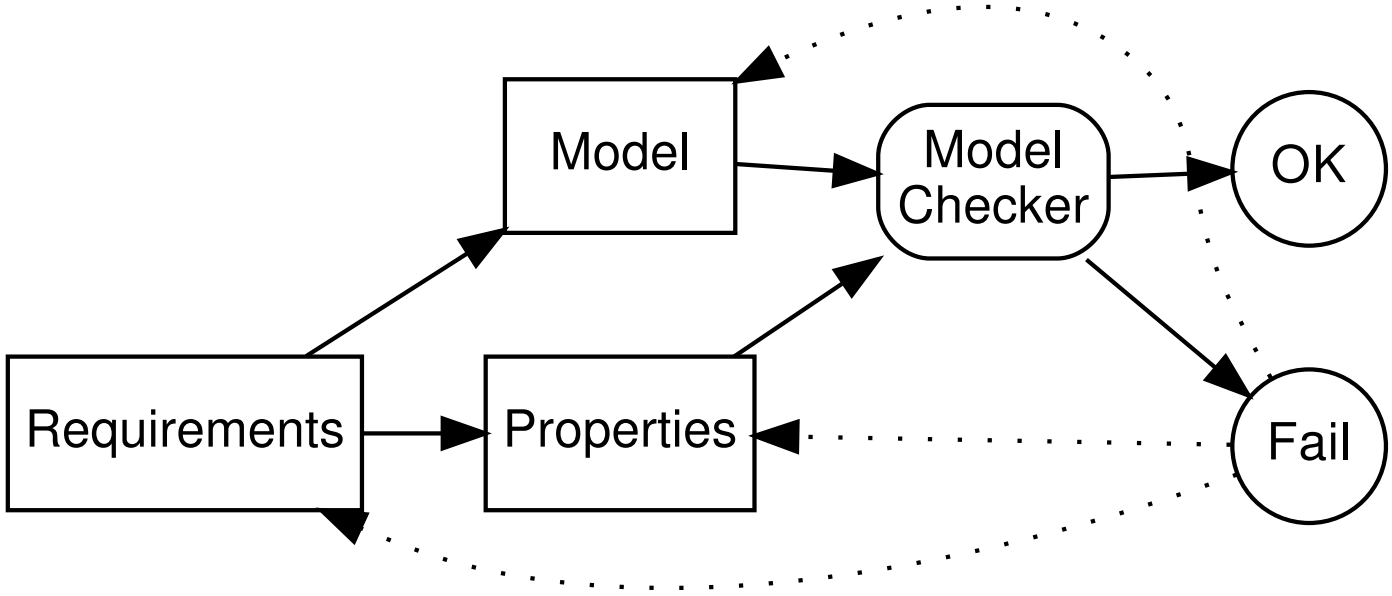
- As a back-end to other tools:
  - → NuSMV-PA: Safety analysis platform
  - → Back-end for Petri net model checking (another modeling approach).
  - → Test case generation.
- Case studies:
  - → Kerberos protocol.
  - → Web service composition.
  - → Railway interlocking control tables.

A critique of NuSMV

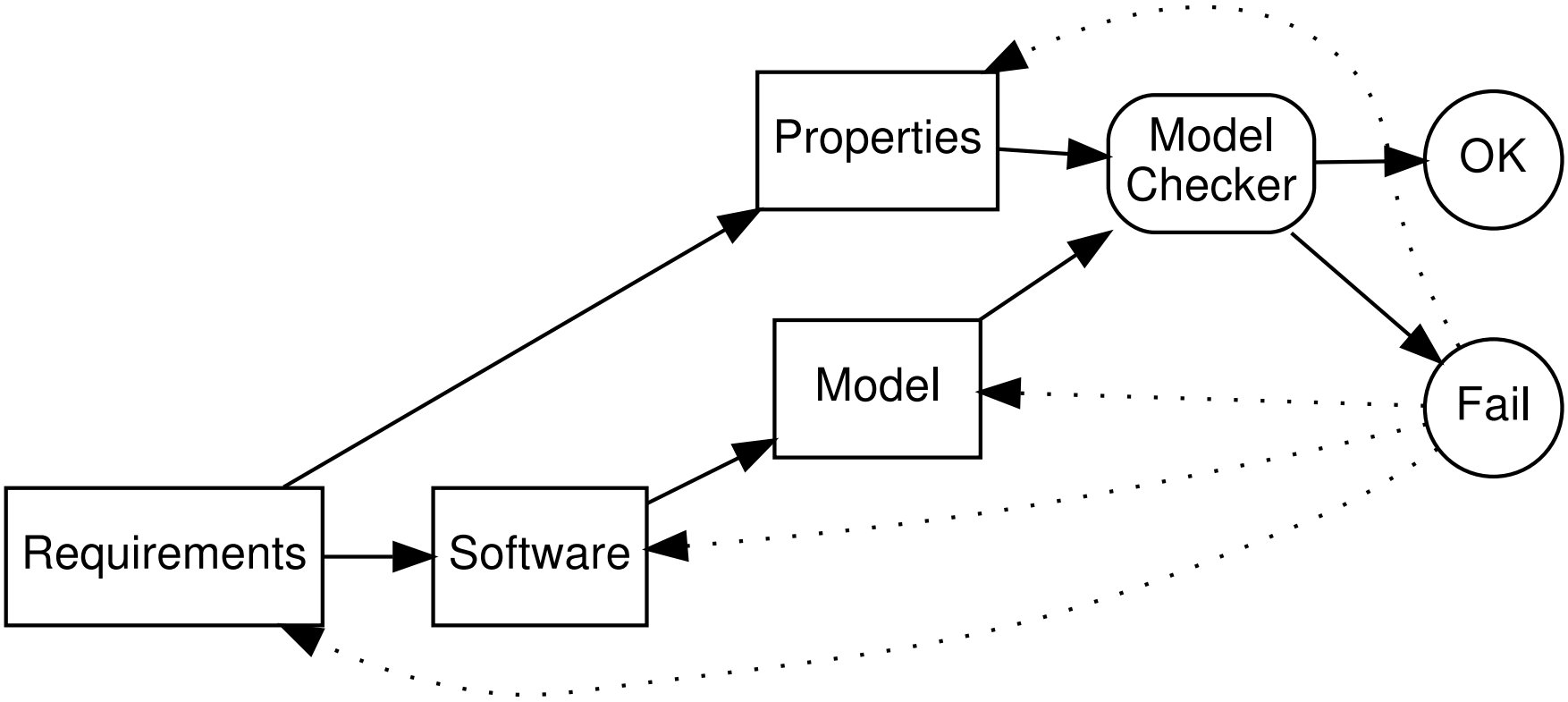
Pro	**Con**
open source	limited open tool set
mature	limited syntax
fast	no arrays of modules
well-defined semantics	focus on synchronous systems

Protocol/algorithm verification

- Knowledge on security/safety/reliability concerns.
- Logics to express temporal properties.
- Tools to verify transition systems.

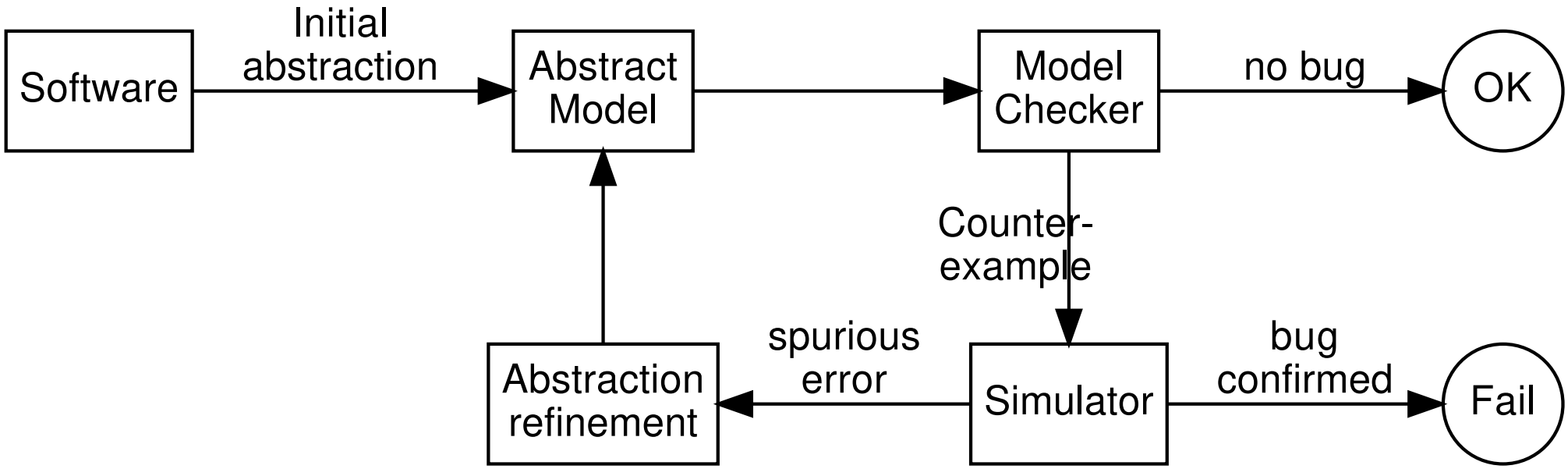


Software verification



Challenging to maintain model by hand.

Counter-Example Guided Abstraction Refinement (CEGAR)



Practical program verification for small systems.

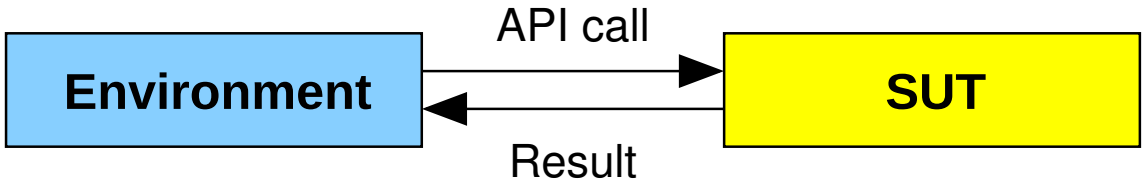
The SLAM Toolkit

- Goal: Verify Windows NT device drivers by model checking.
- System calls approximated by model.

  $\rightarrow$ Model includes state changes in kernel.
- Model used to check dozens of device drivers.
- Continuous effort (over 5 man-years for kernel model alone!)
- For MC a single application, manual abstraction more economical.

Assignment: Read paper on SLAM, answer quiz.

Model-based Testing vs. Model Checking



SUT = System under test; API = Application programming interface

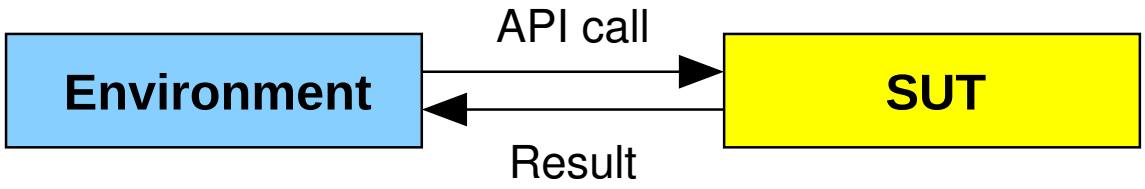
Test model

What

System model

How

Test Model vs. System Model



SUT = System under test; API = Application programming interface

Test model

- Represents **environment**.
- Models system **behavior**.
- Used to generate **test** cases.
- Model, test one module at a time; SUT itself provides counterpart.
- Model-based testing.

System model

- Represents **system** itself.
- Models system **implementation**.
- Used to **verify** system.
- Need model of most components to analyze system behavior.
- Model checking, theorem proving.

Summary

Model checking

Symbolic checking (last week).

- Everything is a bit vector.
- Efficient representation: BDDs.

Bounded model checking.

- Expand k trans. $\rightarrow$ 0-order formula.
- Efficient solution: SAT solver.

Model validation

- Negate properties.
- Check if antecedent (a in $a \rightarrow b$) is ever true.
- Ensure „interesting” states are actually reachable.

Advanced NuSMV exercise: Elevator

Given: Partial elevator model.

Goals:

1. complete the model by formalizing the transition relations,
2. formalize the given properties as temporal-logic properties, and
3. ensure that the requirements are satisfied.

Base features (required to pass)

Buttons

- One button per floor, can be pressed non-deterministically.
- A pressed button stays active until the controller resets it.

Cabin

- Cabin can be at any floor between 1 and 3.
- Controller decides the direction (up/down/stop).

Door

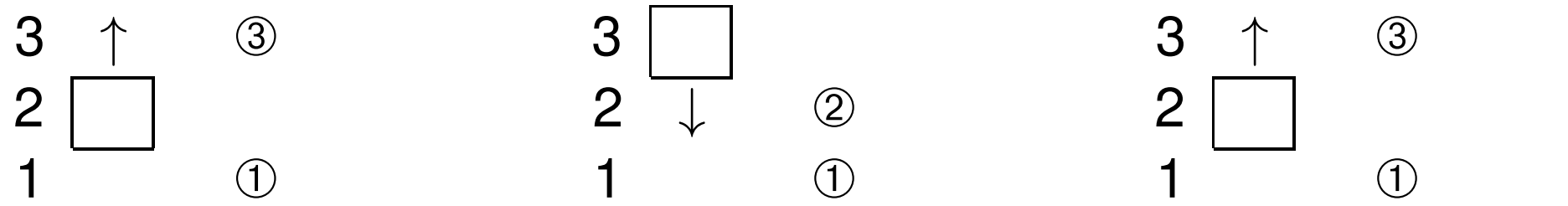
- Door can be open or closed.
- Door responds to commands “open”, “close”, and “nop”.

Controller (also required to pass)

- Moves cabin,
- opens/closes door,
- resets the corresponding button when the elevator has served a request.

Input: current floor,
status of the door,
direction in which the cabin is moving,
status of all buttons.

Avoid „bouncing” (without resorting to fairness constraints)



(a) Elevator stops at floor 3. (b) Elevator stops at floor 2. (c) Elevator goes back to floor 3.

Figure 1: An example where the elevator ends up alternating endlessly between floors 2 and 3.

Properties (required to pass)

- Ten properties (liveness or safety) have to be filled in.
- Properties have to be correct and fulfilled by the model.
- LTL or CTL is fine (if the property is expressable in that logic).

Optional tasks for higher grades

Door safety

The door contains a sensor, which is triggered if something obstructs the door and may physically prevent it from closing.

Earthquake safety

The elevator has a sensor that detects strong shaking. In that case, it stops immediately (at the next floor) and opens the doors

- Each optional task includes model extensions and new or updated properties.
- Both parts have to be correct for the points to be given.
- Optional tasks can be submitted independently and in any order.

Another optional task: directional buttons

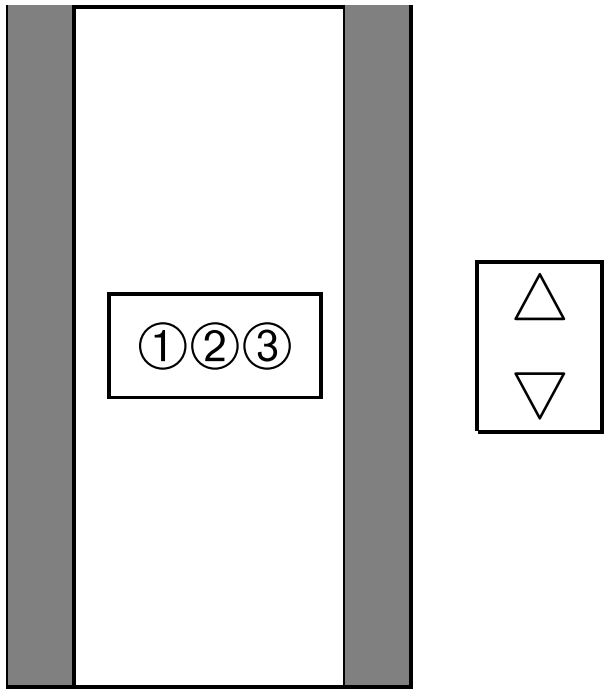
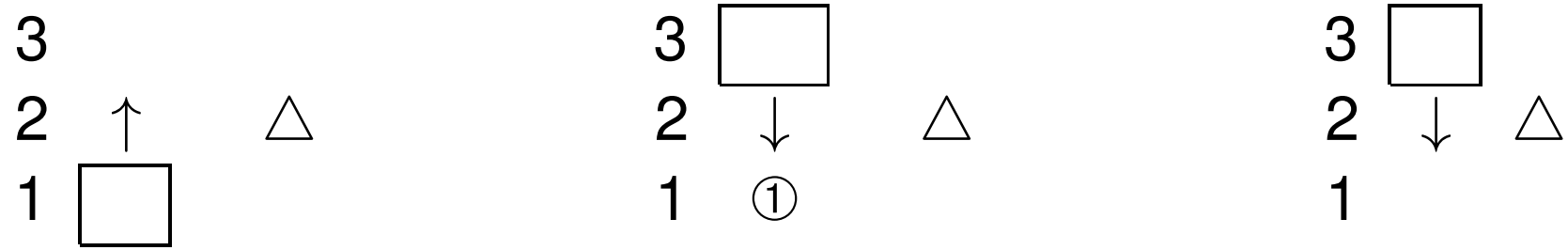


Figure 2: A schematic showing the elevator with open doors, and buttons inside and outside the cabin.

Directional buttons: stop only when needed



(a) Elevator stops at floor 2. (b) Elevator skips floor 2. (c) Elevator stops at floor 2.

Figure 3: Three scenarios with *p_2_up* pressed: elevator moving up; elevator moving down with an active request on floor 1; elevator moving down with no other requests active.

Other optional tasks

Lobby mode

In some hotels, elevators are configured to return to the lobby (in this case, floor 1) when not used.

Validation, documentation

Document error trace(s) with incorrect model and/or property.

Grading criteria

Description	Feature	Property	Points
Door sensor.	easy	easy	+0.5
Earthquake sensor, at least one correct property.	medium	easy	+0.5
All related properties are correct.		medium	+0.25
Directional buttons.	difficult	medium	+1.5
Lobby mode.	easy		+0.25
Property/fairness condition related to lobby mode.		difficult	+0.5
Documented error trace of incorrect model.	easy		+0.25
Documented error trace of incorrect property.		easy	+0.25

Up to half of the points can be submitted individually (in separate assignment).

Late submissions reduce the top attainable grade.